PARALLEL CONVEX HULL

BY SANJAY MUTHU MANICKAM

UNDER THE SUPERVISION OF DR. RUSS MILLER
CONVEX HULL

• The Convex hull of a set of $X$ points that is the smallest convex set that contains $X$.

• It can be solved using the Graham Scan method.
GRAHAM SCAN
CONVEX HULL PARALLEL IMPLEMENTATION
• The points are divided into two halves.

• Compute the convex hull of each subset.

• Now combine the two hulls, this can be done by finding the upper and lower common tangent and removing the points in between the overall outline.

• This gives us the new convex hull coordinates.
PARALLEL IMPLEMENTATION

• Each processor generates random data for the computation of the convex hull.
• Each processor parallelly computes the convex hull of the data generated.
• Now each processor will send its data to the next processor in powers of 2.
• After receiving the data each processors merges the hulls and sends on the new convex hull to the next processor.
MPI COMMANDS USED

- MPI_WTIME
- MPI_SENDRECV
- MPI_TYPE_CREATE_STRUCT
- MPI_COMMIT
RUNNING TIME WITH READING FROM FILES

The graph shows the running times for different numbers of processors and file sizes. The x-axis represents the number of processors, while the y-axis represents the time. Three lines are plotted, each representing different file sizes:
- Blue line for 100000 files
- Orange line for 500000 files
- Green line for 1000000 files

As the number of processors increases, the time decreases for all file sizes, indicating improved performance with more processors.
SPEEDUP WITH READING FROM FILES

Speedup

Processors

- 100000
- 500000
- 1000000

Speedup

5 10 15 20 25 30

4 5 6 7 8 9
RUNNING TIME WITHOUT READING FROM FILES

![Graph showing running times for different data sizes and processor counts.](image)

### Data Size

<table>
<thead>
<tr>
<th></th>
<th>10000</th>
<th>500000</th>
<th>1000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.03085</td>
<td>0.16876</td>
<td>0.31786</td>
</tr>
<tr>
<td>8</td>
<td>0.02608</td>
<td>0.14782</td>
<td>0.28094</td>
</tr>
<tr>
<td>16</td>
<td>0.02179</td>
<td>0.10786</td>
<td>0.20134</td>
</tr>
<tr>
<td>32</td>
<td>0.01542</td>
<td>0.06832</td>
<td>0.14563</td>
</tr>
<tr>
<td>64</td>
<td>0.01325</td>
<td>0.06312</td>
<td>0.12898</td>
</tr>
<tr>
<td>128</td>
<td>0.01783</td>
<td>0.09862</td>
<td>0.1684</td>
</tr>
</tbody>
</table>
SPEEDUP WITHOUT READING FROM FILES.
EFFICIENCY WITHOUT READING FROM FILES
OBSERVATIONS

• Sequential performance was as predicted.

• Parallel Speedup was not true to prediction in all cases.

• There is an ideal no of processors to be used for which we will have the maximum benefit, after which there will be minimal decrease in the time taken for the program to run.
CHALLENGES FACED

• DATA GENERATION.
  • I tried to create my own data and read from the file. It took too long to read from files, so I started generating data in each node of random number's within a given size.

• DEBUGGING PROGRAMS
  • I had to write the data to a file to debug.

• MEMORY LIMITATIONS
  • Sometimes I ran out of memory when using a large number of points.
REFERENCES

- https://www.mpich.org/
THANK YOU